**Webservice**

**1 Web service framework**

Jersey,Spring Rest (MVC)

**2 Scenario Employee id ,NAME sending ID and NAME based on ID update name**

**wt will be the HTTP call PUT r PSOT explain till class level and method level creation not to dao logic**

**can put be replaced by post will it work?**

It has to be PUT method because POST is non idempotent so it will create new resource every time

PUT will check if ID exist it will update resource if not it will create resource

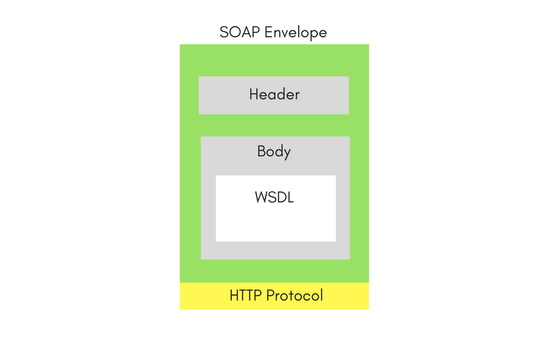
But if I replace with POST it will always try to create a new resource so constraint violations r duplication occurs

**3 SOAP and REST based web service**

## **SOAP Web Services**

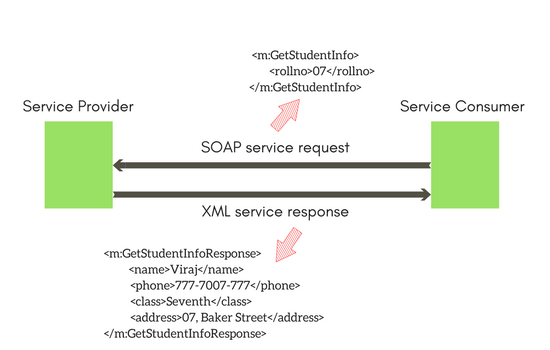
SOAP is an XML-based protocol. The biggest advantage of using the SOAP Web Service is its own security. SOAP stands for Simple Object Access Protocol.

SOAP provides an envelope to send a web services messages over the Internet, using the HTTP protocol. The messages are generally in XML format.



In simple words, SOAP is a technique to send an XML request over the Internet using HTTP protocol (hitting a URL), and in return getting an XML response.

Taking a real world example, if a client wants to fetch a school's student data, by sending in the student's Roll No. in the request, he can do so using web services. But how will the client know, which URL to call and what to send in the request?



Well, every application serving SOAP requests, has a WSDL file. WSDL is an XML, and it stands for Web Service Description Language. WSDL describes all the methods available in the web service, along with the request and response types. It describes the contract between service and client.

SOAP was intended to be a way to do remote procedure calls to remote objects by sending XML over HTTP.

If we look at the current software industry, you will find that, SOAP is being used in the enterprise applications, generally in the legacy code. Today the world is moving fast towards the RESTful Web Services.

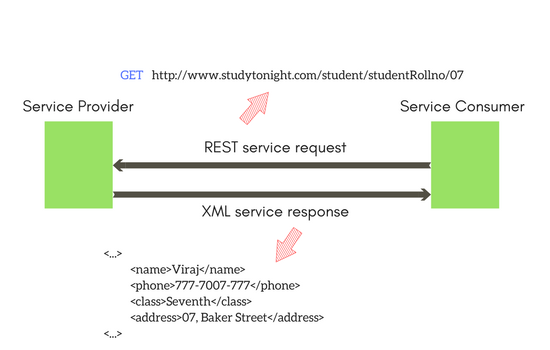
## **REST Web Services**

The REST stands for Representational State Transfer. REST is not a set of standards or rules, rather it is a style of software architecture. The applications which follow this architecture are referred to as RESTful

Unlike SOAP which targets the actions, REST concerns more on the resources. REST locates the resources by using URL and it depends on the type of transport protocol(with HTTP - GET, POST, PUT, DELETE,...) for the actions to be performed on the resources. The REST service locates the resource based on the URL and performs the action based on the transport action verb. It is more of architectural style and conventions based.

For Example: in a RESTful architecture, this URL http://{serverAddress}/students/studentRollno/07 can be used to:

* To get student information by sending a REST call of GET type, and the service will return information of student with roll no as 07
* The same service can also be used to update the student data, by sending in the new values as Form data in a PUT request.



SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON. SOAP cannot make use of REST whereas REST can make use of SOAP.

RESTful Web services are completely stateless. Managing the state of conversation is the complete responsibility of the client itself. The server does not help you with this. Normally, a SOAP Web services are stateless – but you can easily make SOAP API stateful by changing the code on the server.

REST operates through a solitary, consistent interface to access named resources. It’s most commonly used when you’re exposing a public API over the Internet. SOAP, on the other hand, exposes components of application logic as services rather than data.

Additionally, it operates through different interfaces. To put it simply, REST accesses data while SOAP performs operations through a more [standardized set](http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/" \t "https://stackify.com/soap-vs-rest/_blank) of messaging patterns. Still, in most cases, either REST or SOAP could be used to achieve the same outcome (and both are infinitely scalable), with some differences in how you’d configure it.

SOAP is a protocol which means a set of rules. JSON is an object. SOAP can use JSON for communication but the reverse is not at all possible. SOAP uses XML format whereas JSON uses a key-value pair.

REST really only uses the HTTP verbs ( GET , PUT , POST , DELETE , …) on a resource. ... That is why there isn't really a WSDL for a REST service since you only ever have 4 methods on the resource. But you still have the possibility to describe a REST web service with WSDL 2.0.

Differences: REST API has no has no official standard at all because it is an architectural style. SOAP API, on the other hand, has an official standard because it is a protocol. REST APIs uses multiple standards like HTTP, JSON, URL, and XML while SOAP APIs is largely based on HTTP and XML.

## **Benefits of REST Over SOAP**

In addition to using HTTP for simplicity, REST offers a number of other benefits over SOAP:

* REST allows a greater variety of data formats, whereas SOAP only allows XML.
* Coupled with JSON (which typically works better with data and offers faster parsing), REST is generally considered easier to work with.
* Thanks to JSON, REST offers better support for browser clients.
* REST provides superior performance, particularly through caching for information that’s not altered and not dynamic.
* It is the protocol used most often for major services such as Yahoo, Ebay, Amazon, and even Google.
* REST is generally faster and uses less bandwidth. It’s also easier to integrate with existing websites with no need to refactor site infrastructure. This enables developers to work faster rather than spend time rewriting a site from scratch. Instead, they can simply add additional functionality.

## **Benefits of SOAP Over REST**

## Because you can achieve most outcomes using either protocol, it’s sometimes a matter of personal preference. However, there are some use cases that SOAP tends to be better-suited for. For instance, if you need more robust security, SOAP’s support for WS-Security can come in handy. It offers some additional assurances for data privacy and integrity. It also provides support for identity verification through intermediaries rather than just point-to-point, as provided by SSL (which is supported by both SOAP and REST).

## Another advantage of SOAP is that it offers built-in retry logic to compensate for failed communications. REST, on the other hand, doesn’t have a built-in messaging system. If a communication fails, the client has to deal with it by retrying. There’s also no standard set of rules for REST. This means that both parties (the service and the consumer) need to understand both content and context.

## **Other benefits of SOAP include:**

## SOAP’s standard HTTP protocol makes it easier for it to operate across firewalls and proxies [without modifications](http://searchmicroservices.techtarget.com/tip/REST-vs-SOAP-Choosing-the-best-web-service" \t "https://stackify.com/soap-vs-rest/_blank) to the SOAP protocol itself. But, because it uses the complex XML format, it tends to be slower compared to middleware such as ICE and COBRA.

## Additionally, while it’s rarely needed, some use cases require greater transactional reliability than what can be achieved with HTTP (which limits REST in this capacity). If you need ACID-compliant transactions, SOAP is the way to go.

## In some cases, designing SOAP services can actually be less complex compared to REST. For web services that support complex operations, requiring content and context to be maintained, designing a SOAP service requires less coding in the application layer for transactions, security, trust, and other elements.

## SOAP is highly extensible through other protocols and technologies. In addition to WS-Security, SOAP supports WS-Addressing, WS-Coordination, WS-ReliableMessaging, and a host of other web services standards, a full list of which you can find on [W3C](https://www.w3.org/Submission/" \t "https://stackify.com/soap-vs-rest/_blank).

|  |  |
| --- | --- |
| REST | SOAP |
| REST is a style of software architecture. | SOAP is a protocol or a set of standards. |
| REST stands for Representational State Transfer | SOAP stands for Simple Object Access Protocol |
| REST can use SOAP because it is a concept and can use any protocol like HTTP, SOAP etc. | SOAP cannot use REST because it itself is a protocol. |
| REST uses URI to expose business logic. But as REST works on the basis of type of HTTP request, hence same URI can work for more than a single type of operation. | SOAP uses the service interface to expose business logic. |
| REST does not define too much standards. REST is cool! | SOAP defines standards to be strictly followed. |
| REST inherits security measures from the underlying transport protocols. | SOAP defines its own security layer. |
| REST accepts different data formats like, Plain Text, HTML, JSON, XML etc. | SOAP only works with XML format. |

**4 RESTFULL web service how we create**

* defining the resources accessible via HTTP
* identifying such resources with URLs
* mapping the CRUD (Create, Retrieve, Update, Delete) operations on these resources to the standard HTTP methods (POST, GET, PUT, DELETE)

**5 security wise which will be better in SOAP and REST**

SOAP is still used in many big organisations. With built-in security and reliability functions, SOAP is a great choice for applications where security is more critical than performance. SOAP is highly extensible.

**6 Data Format in web service**

XML is the data format used to contain the data and provide metadata around it, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI lists what services are available. A Web service is a method of communication between two electronic devices over a network.

The key ingredients, including Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI), have been adopted across the entire software industry.

REST accepts different data formats like, Plain Text, HTML, JSON, XML etc.

**7 JSON to java object dependency how**

GSON , fasterxml.jackson dependency etc

**8 what are RESFULL service and implementation for RESRFUL service (JAX-RX)**

REST stands for Representational State Transfer. It basically leverages the HTTP protocol and model to provide data services. What REST should be is follow a client/server model, be stateless, cacheable, allow for a layered model and provide a uniform interface. This would refer to the user who has an identity of 1 .

* defining the resources accessible via HTTP
* identifying such resources with URLs
* mapping the CRUD (Create, Retrieve, Update, Delete) operations on these resources to the standard HTTP methods (POST, GET, PUT, DELETE)

**9 METHOS of REST (GET,POST,PUT,DELETE) when to use it**

POST - create an new resource

GET- READ /SERACH reosurce

PUT- NEw or Update

DELETE - remove

PATCH- custom update

**10 HTTP status (200,500etc )**

200 - success

1. Bad request
2. Not authorized
3. Not supported method
4. Server internal Exception

11 JSON converted into XML

**12 java library to implement RESTFULL**

Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services. JAX-RS is part of JDK, so you don’t need to include anything to use it’s annotations.

**JAX-RS is the Java API for REST web services whereas JAX-WS is the Java API for SOAP web services.**

There are two major implementations of JAX-RS API.

1. **Jersey**: [Jersey](https://jersey.github.io/" \t "https://www.journaldev.com/9170/_blank) is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.
2. **RESTEasy**: [RESTEasy](https://resteasy.github.io/" \t "https://www.journaldev.com/9170/_blank) is the JBoss project that provides JAX-RS implementation.

**13 How do you validate your user**

1 Based on Basic Authentication credential

2 Token based access

2 oauth based access

**14 how do you test your web services**

Web Services Testing is a type of software testing that validates Web services. The purpose of Web Services Testing is to check the functionality, reliability, performance, and security of an API(Application Program Interface). Web Service Testing is similar to unit testing in some cases. You can test a Webservice manually or create your own automation code or use an off-the shelf automation tool like **Postman**.

**15 PUT and POST method diff and its vice versa**

POST is not idempotent new resource will be created PUT is idempotent it will be unchangeable for multiple call

POST is server controlled where PUT is client controlled

Add new Resource-POST

Update a resource-PUT

Client Generate Id based add resource=PUT

**PATCH VS PUT**

PUT is for full update

PATCH is for customized update to update only mentioned resource

**16 idempotent and non idempotent(Safe method)**

Safe methods are HTTP methods that do not modify resources. For instance, using GET or HEAD on a resource URL, should NEVER change the resource. However, this is not completely true. It means: it won't change the resource representation. It is still possible, that safe methods do change things on a server or resource, but this should not reflect in a different representation.

This means the following is incorrect, if this would actually delete the blogpost:

GET /blog/1234/delete HTTP/1.1

Safe methods are methods that can be cached, prefetched without any repercussions to the resource.

An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. Again, this only applies to the result, not the resource itself. This still can be manipulated (like an update-timestamp, provided this information is not shared in the (current) resource representation.

Consider the following examples:

1 a = 4;

2 a++;

The first example is idempotent: no matter how many times we execute this statement, a will always be 4. The second example is not idempotent. Executing this 10 times will result in a different outcome as when running 5 times. Since both examples are changing the value of a, both are non-safe methods.

Idempotency is important in building a fault-tolerant API. Suppose a client wants to update a resource through POST. Since POST is not a idempotent method, calling it multiple times can result in wrong updates. What would happen if you sent out the POST request to the server, but you get a timeout. Is the resource actually updated? Does the timeout happened during sending the request to the server, or the response to the client? Can we safely retry again, or do we need to figure out first what has happened with the resource? By using idempotent methods, we do not have to answer this question, but we can safely resend the request until we actually get a response back from the server.

Be careful when dealing with safe methods as well: if a seemingly safe method like GET will change a resource, it might be possible that any middleware client proxy systems between you and the server, will cache this response. Another client who wants to change this resource through the same URL (like: <http://example.org/api/article/1234/delete>), will not call the server, but return the information directly from the cache. Non-safe (and non-idempotent) methods will never be cached by any middleware proxies

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Method | Idempotent | Safe |  |
| OPTIONS | yes | yes |  |
| GET | yes | yes | Call request many times it unchangeable only provides the result |
| HEAD | yes | yes |  |
| PUT | yes | no | Calling mutiple time its already updated for the 1st execution so mutiple time called fro same case it is unchangeable |
| POST | no | no | Call multiple times those many time objects created impact on performance so its non idempotent |
| DELETE | yes | no | Mutiple delete already deleted so not gona impact system unchangable |
| PATCH | no | no |  |
|  |  |  |  |

17 y to go for licensed web servers when tom cat is open source

**18 how to perform secure transmission**

Security is critical to web services. However, neither XML-RPC nor SOAP specifications make any explicit security or authentication requirements.

One of the security measures available for the HTTP is the HTTPS protocol. HTTPS is the secure way of communication between the client and the server over the web. HTTPS makes use of the Secure Sockets layer or SSL for secure communication.

Ten ways to secure Web services

Secure the transport layer. ...

Implement XML filtering. ...

Mask internal resources. ...

Protect against XML denial-of-service attacks. ...

Validate all messages. ...

Transform all messages. ...

Sign all messages. ...

Timestamp all messages.

With RESTful web services, SSL authentication is slightly different than other SSL authentications. An SSL certificate, which assures authentication, is a standard technology ensuring web interactions are secure. The process of securing a website starts when a user accesses a site.

Best Practices to Secure REST APIs

Keep it Simple. Secure an API/System – just how secure it needs to be. ...

Always Use HTTPS. ...

Use Password Hash. ...

Never expose information on URLs. ...

Consider OAuth. ...

Consider Adding Timestamp in Request. ...

Input Parameter Validation.

**MICROSERVICE**

**1 MICROSERVICE EXP and years**

1 year

**2 A/c of micro-service in last project**

Video Wall with Payments and Ledger part and meter details Master account

3 **coding to micro service**

register with eureka service register (eureka server and client) and Hystrix to check micro service health.

Creating micro service instead of using IP can use eureka register server and make all microservice to register itself eureka server

Implementing Circuit breaker using Hystix using this we can monitor thread request for those service fails and success case based on that we can provide circuit breaker configuration and timeout time call back method in case is server is done are taking to much time. Using Hysrix dashbaord we can trace those request status of reg micro service

API gate Way so req will be redirected from the API gateway

Cloud Config Server where all generic configuration kept and used in application using bootstrap.yml file

Zipkin and sleuth for tracing the logs from microservice it keep transaction if and span id so it is trace the dependency

Elastic search for logs storage

**4 Security in micro service**

* Encrypt all communications (using https or transport layer security).
* Authenticate all access requests.(JWT token based)
* Do not hard code certificates, passwords or any form of secrets within the code.
* Use DevSecOps tools designed for microservice architecture environments to scan code as it is developed.
* Define the APIs and strictly make sure all communications comply.

**5 one is calling two and two is calling three**

(this is wrong bcz it is not sequenced )if it dependent

**6 Business phase API registry maintained**

We need to register Business APi in order to maintain eureka discovery and client we are using to register itself with all multiple instance can be handled using this and their health can be monitored

**7 difficulties in micro-service disadvantage**

| **Pros** | **Cons** |  |  |  |
| --- | --- | --- | --- | --- |
| Greater agility | Needs more collaboration (each team has to cover the whole microservice lifecycle) |  |  |  |
| Faster time to market | Harder to test and [monitor](https://raygun.com/blog/monitoring-microservices/" \t "https://raygun.com/blog/what-are-microservices/_blank) because of the complexity of the architecture |  |  |  |
| Better scalability | Poorer performance, as microservices need to communicate (network latency, message processing, etc.) |  |  |  |
| Faster development cycles (easier deployment and debugging) | Harder to maintain the network (has less fault tolerance, needs more load balancing, etc.) |  |  |  |
| Easier to create a CI/CD pipeline for single-responsibility services | Doesn’t work without the proper corporate culture ([DevOps culture](https://raygun.com/blog/what-is-devops/" \t "https://raygun.com/blog/what-are-microservices/_blank), automation practices, etc.) |  |  |  |
| Isolated services have better fault tolerance | Security issues (harder to maintain transaction safety, distributed communication goes wrong more likely, etc.) |  |  |  |
| Platform- and language agnostic services | Difficult to debug and hard to find where fault is |  |  |  |
| Cloud-readiness | Network call ad complexity, hard to implement network calls service discovery and all |  |  |  |

**8 Transaction working in Micro service**

Distributed transaction is required since each micro-service will be pointing to different db instance

Taking Ex of Shopping cart

- one for balance update of Wallet and for one for order place in any case of one transitional fail it needs to roll back in both in monolithic it was easy but not in micro-service

Two approaches to handle it

1 Two Phase commit

It will have prepare and commit it is having coordinator it will have transaction id sending req to 2 micro service once it got response it will make prepare not gonna commit once both got response then it will commit so issue will be resolved

Three phase commit is extended part of two phase commit with pre commit option

2 SAGA A/c

It has 2 approaches

1 Choreography(Event Based) - using queue system to trigger the events

2 Orchestration Based - using Orchestration service to coordinate in between